Weckstén, M., J. Vasell, and M. Jonsson, "A tool for derivation of implementation constraints – evaluation using implementation simulation," RTSS 2004 WIP Proceedings, The 25th IEEE International Real-Time Systems Symposium, Lisbon, Portugal, Dec. 5-8, 2004.

A Tool for Derivation of Implementation Constraints — Evaluation using Implementation Simulation

Mattias Weckstén, Jonas Vasell and Magnus Jonsson
Centre for Research on Embedded Systems (CERES)
Halmstad University, Halmstad, Sweden
Mattias.Wecksten@ide.hh.se, Jonas.Vasell@ide.hh.se, Magnus.Jonsson@ide.hh.se

***Abstract*— The industrial use of ad hoc implementation methods for non-functional constrained tasks has sometimes resulted in unnecessary expensive projects. In some cases, ad hoc methods result in overly many iterations to be made and in some severe cases, total project breakdown. To be able to solve these problems a new method has been developed to derive end-to-end non-functional constraints, such as performance or resource utilization requirements, to task-level constraints and to promote this information to the implementation phase of the project.**

For a tool to be really useful it is important to be able to show the usability and potential cost reduction. To be able to show that a certain implementation method costs less in work hours than to use an ad hoc implementation method, a model for implementation simulation has been developed. As far as we know, no similar experiments has been done to compare implementation methods.

I. Introduction

Developing a computer system is a very complex task, typically there is a large set of constraints, interacting in ways hard to predict. This alone makes the process of computer system design very hard, even if only fully implemented software components were used. In the typical case, a few components may already be implemented, evaluated and documented (i.e. bought or reused), but most of the components will initially only exist as specifications.

For modern embedded real-time systems, complexity depends on functional requirements (e.g. the brake functionality of a car) as well as non functional requirements (e.g. the maximum allowed jitter between the different brake circuits) [1]. Since most commercially available system development tools only handle functional constraints and as the complexity of non functional constraints steadily increases, other types of development methods are needed. Related research shows that the lack of validation of non functional constraints will be very expensive in the long run [2]. It is cheaper to detect potentially volatile designs directly than to detect them after implementation, since the decision can be made at the source of the problem – whether to abort the project or start over. Note that iterative development is probably unavoidable (i.e. all assumptions will not be correct from start), but design decisions leading to dead ends (or more trouble) should be avoided.

Besides the complexity issue, the design of software components and the dimensioning of the resources are done at a point in the project when resource limitations may be unknown and very little is implemented [3]. A method is thus needed to guide the designer, to keep within the specified non functional constraints, and validate the design, guaranteeing that there will be a way to implement and execute the system [4]. Since many parameters are unknown until implementation is completed, we would like to keep the design flexible for as long as possible to avoid dead ends in the development [5] [6] [7].

This paper presents a method that makes it possible to compare different methods for guiding implementation in terms of implementation cost. This paper also presents the results of an experiment using the proposed evaluation method.

II. The budget generation method

The problem is to find a method that, based on a functional system description with end-to-end constraints and a given platform, generate a set of task level implementation time constraints (ITCs). The functional system description is a directed acyclic task graph, where the designer has estimated the execution time for each task in the graph, in relative terms. Each task can have optional timing constraints (i.e., in the form of release time and/or deadline) and optional locality constraints limiting the possible processor allocations. For each precedence constraint of a task, an optional communication package of estimated relative length can be defined. The platform is modelled as a network with homogeneous processors connected to a single bus. The output from the method is a set of budgets where each budgets in the set consists of ITCs for each task and the associated communication tasks. It is assumed that the deadline for a task is shorter than the task's period.

Here follows an overview of the method illustrated in Fig. 1, further described below. Initial data formatting is done by the preprocessor that generates constrained task graphs, based on the input data (resource parameters, performance restrictions, and task set). The generation of schedules (scheduler in the figure) for the constrained task graph consists of two steps, ordering and allocation. The purpose of the ordering step is to generate execution orders of the tasks according to the precedence constraints in the task graph. As part of the allocation, communication tasks are generated. The allocation of the tasks in the task graph is quite straightforward, generating allowed

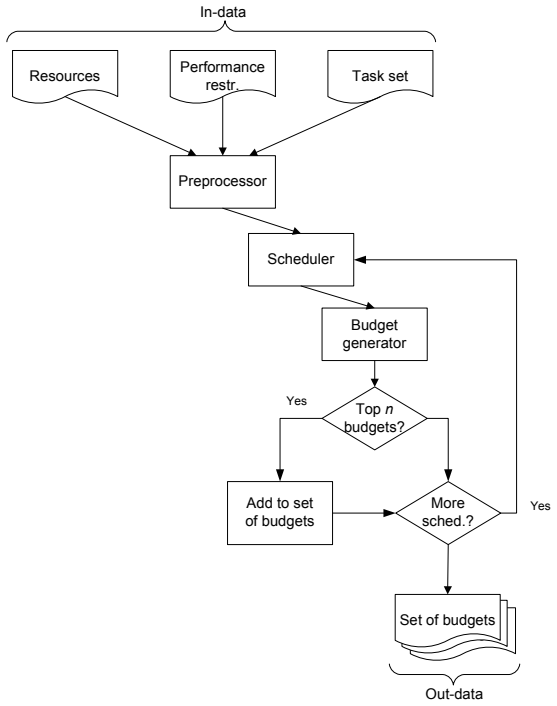

Fig. 1. Flowchart for the suggested implementation of the method. The schedules are turned into budgets and the n best are kept.

allocations according to the locality constraints. When allocation is decided, communication tasks can be generated based on the information about which tasks that have to communicate using the bus. Finally, ITCs (budgets) are generated based on the scheduled task graphs. The top n budgets with minimal tightnesses are saved in the set of budgets that is the out-data from the budget generation method.

To calculate the optimization metric, the budget tightness T, for the scheduled task graph, G (of tasks, V, and precedence constraints, E) the tightness for each path, $p \in P$, in the graph has to be found. As mentioned earlier, for each task there is an estimated execution time V_e. For (at least) the first and last task on the path, there is also a corresponding offset V_o and deadline V_d, respectively. The tightness for a path $p_T, p \in P$ is equal to the fraction between the work W, that is the sum of the task's estimates, and the path length L, that is the difference between the path's deadline and offset. The path's offset is equal to the first task's offset; while the path's deadline is equal to the last task's deadline. In summary we have the following definitions:

$$G = < V, E > \quad (1)$$
$$W = \sum_{v \in p} v_e \quad (2)$$
$$L = p_d - p_o \quad (3)$$
$$p_T = \frac{W}{L} \quad (4)$$
$$T = \max(p_T : p \in P) \quad (5)$$

When the tightest path has been found, the ITCs for the tasks on this path can be generated. The sum of the ITCs for the tasks on the tightest path shall fill out the whole length L of the path. This is done by, for each task on the path, dividing the estimated execution time, v_e, with the path tightness, p_T, giving us the allowed execution time (AET):

$$v_{AET} = \frac{v_e}{p_T} \quad (6)$$

The tasks on the tightest path may now be removed since they all have been given implicit offsets and deadlines (to be used as implementation budgets). One way to calculate the offset and the deadline explicitly for a task on the tightest path is to set the offset for a task to the previous task's deadline, and the deadline to its task's offset plus its AET:

$$v_{i_o} = v_{(i-1)_d} \quad (7)$$
$$v_{i_d} = v_{i_o} + v_{i_{AET}} \quad (8)$$

Since all tasks on the tightest path have explicit offsets and deadlines, it is possible to remove them from the task graph, replacing them with offsets and deadlines. This means that all parents of the task we are going to remove will get new deadlines equal to the removed task's offset. Similarly, all the children of the task we are going to remove will get new offsets equal to the removed task's deadline. When all tasks on the tightest path have been removed from the task graph, the second tightest path is calculated and the corresponding AETs generated. This procedure is then repeated until all tasks have been assigned AETs. Note that only the first tightest path's tightness is of interest when describing the tightness of the whole task graph budget. The consecutive paths' tightnesses are just calculated to be able to generate all tasks' AETs. Although the implementation space (success probability) is maximized in some sense, using the global tightness as optimization metric, there is no guarantee that the implementation will be possible. To avoid the need to start the whole budgeting process over, just because a certain budget did not hold, multiple promising budgets are generated and kept for later use.

Extensions of the method, an example of how the method is used and evaluation of the optimality are found in [8].

III. Usability evaluation method

Although the practicality of the tool has been proven in recent work [8] there is no guarantee that the implementation guidelines reduce the implementation cost. To evaluate this in real life, for real projects, would be good but also quite resource demanding. To get an estimate of wether the budget-based method is better than another given implementation method we have, instead, chosen to simulate the implementation procedure. Simulation makes it possible to get an extensive statistical material and the possibility to repeat experiments for different implementation methods.

The simulation model used for the implementation simulator (see Fig. 2) starts with information about the tasks from the design (TD). This information is used as input to the budgeting algorithm (Bx) which in turn generates

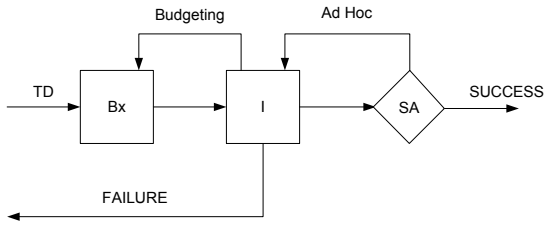

Fig. 2. The model for the implementation simulation.

implementation-time budgets. Bx can, for example, be represented by our budgeting method or an *ad hoc* method, as explained below. If no budgets could be generated this will be considered as a failure of implementation of the pending design and re-design or re-specification may then be needed. Available budgets are forwarded to the (simulated) implementation (I) where WCETs and the accumulated cost for implementing the tasks is calculated. Implementations are evaluated by a schedulability analysis (SA) tool to decide wether it is possible to create an executable schedule for the set of implemented tasks. If this is the case the simulation will terminate positively (SUCCESS) signalling that there exists a schedulable implementation at a certain implementation cost. If a complete system implementation does not exists according to the schedulability analysis we have to reiterate, modifying the budgeting parameters (e.g. updating estimates). If no changes (no new implementations) has been made since the last budget generation, further budget generation is pointless and the simulation will terminate negatively, signaling FAILURE.

When the budget-based implementation method is to be evaluated the budgeting method Bx is represented by the method described in Section II is used. The budgeting algorithm generates several different budgets to avoid iterations if implementation according to some of the budgets fail. If no budgets are generated, the simulator will report an implementation FAILURE. Otherwise, implementation will be simulated by implementing task by task in order of decreasing tightness. After each task implementation, invalid budgets are removed and the process is repeated. When we run out of budgets, or all tasks are implemented, the implementation will be evaluated for schedulability. However, if an implementation is completed according to any given budget we also know that there exists at least one schedule for the implemented tasks and SUCCESS will be reported. Otherwise, the information about the partial implementation will be used to generate a new set of budgets that may make implementation possible.

When implementation according to the *ad hoc* method is evaluated the initial budgeting method Bx is a method generating infinite budgets for all tasks. This means that all possible implementations are valid during implementation, which also means that we know that we will never report a FAILURE since we never run out of budgets. Implementation is performed in the same way as described earlier, but this time the schedulability analysis is needed since we do not know, yet, if we have found a schedulable implementation. If the implemented task set is schedulable SUCCESS will be reported. Otherwise, further implementations have to be done. If adjustments are not possible, FAILURE will be reported.

IV. Simulation model and evaluation

It is assumed that the implementation of a single task starts with a coarse proposal that is refined during the implementation into the final implemented component. To model this in simulation, it is assumed that all tasks have an estimate of the execution time for the component to be implemented (EWCET) and an actual worst case execution time (AWCET). The AWCET is the WCET for the implemented component and is unknown until implementation has been completed. During implementation, the cost in work hours is accumulated to give the total number of work hours needed to complete the project.

The proposed implementation simulator method allows several implementation alternatives for a task, with different AWCET and cost, making incremental implementation possible. However, in this paper only one implementation alternative per task is used.

For each task, a random AWCET is generated from a rectangular distribution centered at the EWCET, where the width of this window is proportional to the certainty of the estimate. To avoid trivial implementations, a constant can be added to the EWCET to shift the distribution, resulting in higher AWCETs and therefore generating implementation simulations of increased difficulty (i.e. introducing an offset for optimistic estimates). The AWCETs in the experiments are generatedS from a distribution that is 1.5 times the estimate in width, and with an offset that sets the lowest AWCET to 5/8 of the estimate. For example, for a task with an estimate of 100, the offset $AWCET_o$ is 62.5 and the width of the distribution is 150, meaning that the AWCET will be between 62.5 and 212.5 (i.e. $AWCET_o$ and $AWCET_o + 1.5e$, respectively).

The cost for a task's implementation is considered as a function inverse proportional to the AWCET, where a lower AWCET leads to higher cost. This is implemented in the experiments as the estimate, e, divided by the AWCET minus the offset $AWCET_o$, plus a constant, c (to avoid division by zero):

$$cost = \frac{e}{AWCET - AWCET_o + c} \tag{9}$$

This is based on the assumption that the cost for implementing a certain task will be higher the closer to the $AWCET$ limit ($AWCET_o$) the implementation gets. The described cost function works for the single implementation alternative evaluation used in this paper. For incremental implementations, however, a more detailed cost function will be needed.

A. Evaluation

The evaluation has been made for the special case where only one implementation alternative per task is used. In this case, where each task has only a single implementation alternative, it is easy to calculate the cost

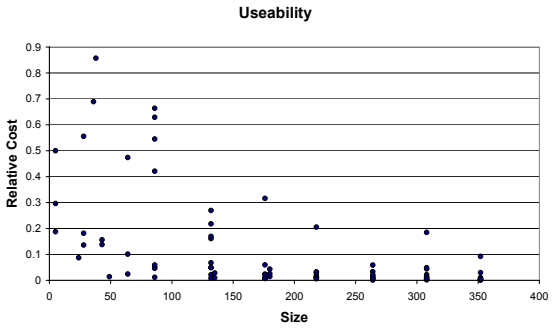

Fig. 3. The relative cost drops with the task graph size in favor of the budget based method.

of the *ad hoc* method since the cost for this special case is equal to the sum of the cost for the implementation of all tasks. Schedulability is evaluated and FAILURE or SUCCESS is reported. For the budget-based method, tasks must be implemented until all tasks are implemented or all generated budgets are violated. Estimates and uncertainties are updated for the implemented tasks and new budgets are generated. If no further implementations are possible, although new budgets are generated, FAILURE is signaled. If all tasks are successfully implemented, SUCCESS is reported.

The budget-based implementation method has been compared to the *ad hoc* implementation method. The methods were evaluated using the same set of applications and the same sets of budgets for each application. The applications are a mix of handmade applications and applications generated by random, ranging in size from five to 80 tasks. The evaluation shows that the budget-based implementation costs less in terms of implementation work than the *ad hoc* implementation. This means that, in the case where implementation is impossible, the budget-based method indicates this at an early stage in the implementation (corresponding to a low cost), while the *ad hoc* method have to implement all tasks to even be able to evaluate the implementation success (more costly). Note that, for the single implementation alternative experiments, the choice of cost function will only affect the total cost for an implementation, not whether it will lead to a FAILURE or not.

B. Discussion

In the case of implementation success, the budget based implementation method and the *ad hoc* implementation method will generated the same cost. This is because there are only a single implementation alternative and all tasks have to be implemented to generate a successful implementation.

In the cases of implementation failure, the budgeting method identifies this at a lower cost in work hours than the *ad hoc* method (see Fig. 3).

These results clearly point out that the proposed budget-based method will be cheaper when it is needed most – when the implementation is leading to a dead end. The proposed method should be extended further, allowing better modelling of implementation cost and supporting incremental implementations. It would also be interesting to evaluate other popular implementation methods for comparisons.

To make the implementation simulation more realistic, a possible extension would be to allow several possible implementations alternatives and not just one. One possibility would be to assign initial costs and AWCETs (e.g. according to the single alternative method described earlier) and extend this with the possibility to make minor adjustments at a low cost. One way would be to model incremental implementation as a function in which an input of the desired AWCET results in an AWCET and an associated cost. The cost could be exponentially proportional to the difference between the old AWCET and the desired AWCET. Incremental implementation would be possible down to a predetermined level of AWCET, below which the cost becomes infinite.

V. Conclusions

In this paper we have argued for the practicality of a tool that helps designers and implementers of embedded systems to make early judgements about the possibilities to meet the system's non-functional constraints, thus making it possible to reduce the costs and risks in system development. A method for evaluating the cost using different implementation methods has been presented. Experiments using the evaluation method has showed the potential of reducing implementation cost using the budget based method.

Future work will include the development of implementation simulation models for other commonly used development methods.

References

[1] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, and N. Bånkestad, "Experiences from introducing state-of-the-art real-time techniques in the automotive industry," in *Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. IEEE, Apr. 2001, pp. 111–118.

[2] D. C. Gause and G. M. Weinberg, *Exploring requirements: quality before design*. Dorset House, 1989, ch. 2, pp. 17–18.

[3] J. Plantin and E. Stoy, "Aspects of system-level design," in *Proc. 7th International Workshop on Hardware/Software Codesign*. ACM Press, 1999, pp. 209–210.

[4] A. Dasdan and R. Gupta, "Timing issues in system-level design," 1998. [Online]. Available: citeseer.nj.nec.com/article/dasdan98timing.html

[5] G. Le Lann, "A methodology for designing and dimensioning critical complex computing systems," in *Proc. IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, Mar. 1996, pp. 332–339.

[6] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *Software Engineering*, vol. 21, no. 7, pp. 579–592, 1995. [Online]. Available: citeseer.nj.nec.com/gerber95guaranteeing.html

[7] D. Sciuto, F. Salice, L. Pomante, and W. Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *Tenth International Workshop on Hardware/Software Codesign*. ACM SIGDA, May 2002, pp. 55–60.

[8] M. Wecksten, J. Vasell, and M. Jonsson, "Towards a tool for derivation of implementation constraints," in *Proc. International Conference on Engineering of Complex Computer Systems*. IEEE, Apr. 2004, pp. 119–127.